# Trading off consistency and efficiency in version-space induction*

*CLAUDIO CARPINETO*
Fondazione Ugo Bordoni
Via Baldassarre Castiglione 59, 00142 - Rome
E-mail: fubdpt5@itcaspur.bitnet

## Abstract

The Candidate Elimination (CE) algorithm, like most inductive learning algorithms, uses a fixed restricted concept language to focus the search on useful generalisations. The drawback of this approach is that it can easily give rise to inconsistency with data, while not ensuring tractability of the search involved. A more flexible way to trade-off efficiency and consistency during concept learning is to work with a variable-sized concept language, starting with small sizes - more efficient and less consistent - and shifting to larger sizes - more consistent and less efficient - when necessary. In this paper we propose an algorithm, called Factored Candidate-Elimination (FCE) algorithm, for inducing version spaces over a set of variable-factored conjunctive concept languages. FCE employs the standard CE algorithm to do induction within each concept language, but it is then able to induce the new version spaces after any language shift without reprocessing the instances already seen. We discuss two applications of this framework, i.e. inducing consistent version spaces when a set of concept languages inconsistent with data is initially available, and inducing version spaces more efficiently when the initial concept language is consistent with data and can be factored. We evaluate the latter with respect to a tree-structured attribute-based conjunctive concept language, and we show when this approach leads to a reduction in complexity.

## 1. Introduction

The Candidate Elimination (CE) algorithm is the best known exemplar of the generalization-as-search paradigm [Mitc82]. It incrementally learns concepts from positive and negative instances performing a bidirectional search through the space of hypotheses described by the concept language. The CE algorithm takes advantage of the partial order

* Work carried out within the framework of the agreement between the Italian PT Administration and the Fondazione Ugo Bordoni

defined by generality on such hypotheses : it represents and updates the set of all concepts that are consistent with data (i.e. the version space) by maintaining two boundary sets, the set S containing the maximally specific consistent concepts and the set G containing the maximally general consistent concepts. The procedure to update the version space is as follows. A positive example prunes concepts in G which do not cover it and causes all concepts in S which do not cover the example to be generalized just enough to cover it. A negative example prunes concepts in S that cover it and causes all concepts in G that cover the example to be specialized just enough to exclude it. As more examples are seen, the version space shrinks, possibly converging to a single target concept.

This approach has two main shortcomings. The first is that since practical applications of the CE algorithm require a restricted concept language, it may be unable to induce consistent concepts. The second is that even with a restricted concept language it usually lacks computational efficiency. To illustrate this point, consider the simple but common case of a conjunctive concept language defined on a tree-structured attribute-based instance space. If there are k attributes, and each attribute tree has l levels and branching factor b (hence $b^{l-1}$ leaves and $b^l - 1$ values in each tree), there are in all $(b^{l-1})^k$ instances. While the total number of concepts that can be expressed with such instances is $2^{(b^{l-1})^k}$, the concept language contains only $(b^l - 1)^k$ concepts. On the other hand, the same restricted concept language does not guarantee tractability of the CE algorithm, in that it has been shown that although the set S stays stable [Bund85] the set G may grow exponentially due to its fragmentation [Haus88].

By modifying the basic algorithm and/or introducing formalized knowledge recent research has addressed both the consistency [Utgo86, Hirs90] and the efficiency issue [Mitc82b, Hirs89, Rose90, Carp91, Nico91]. However these two problems have been dealt with in isolation, whereas they are strictly connected. In fact, both vary with the size of the concept language, yet in two inverse ways. A large concept language is likely to produce consistent concepts in an inefficient manner, a small concept language is likely to be efficient at cost of consistency[1]. As long as we use a fixed concept language there seems to be no way of escaping this kind of dilemma. A more flexible strategy is to use a set of concept languages, and have the version space approach choose the one that best trades off consistency and efficiency for a given training set. In this paper we investigate this idea, providing a framework for inducing version spaces over a set of variable-factored conjunctive concept languages and discussing its applications to improve consistency and reduce complexity. The rest of the paper is organized as follows. We first define the learning problem. Then we present an algorithm, the Factored Candidate-Elimination (FCE) algorithm, to solve the learning problem efficiently. Next we evaluate this approach;

---

[1]This is an approximation. Complexity of the CE algorithm depends on the square of the boundary sets' size [Mitc82a]; that is, on the presence of large sets of unordered elements in the concept language.

in particular, we compare the CE and FCE algorithms' complexity for a tree-structured attribute-based conjunctive concept language showing when the latter is better.

## 2. Definition of the learning problem

We first introduce the notions that characterize our learning problem. In the following concepts are viewed as sets of instances and languages as sets of concepts.

A concept $c_1$ is *more general than* a concept $c_2$ if the set of instances covered by $c_1$ is a proper superset of the set of instances covered by $c_2$. We shall not make any particular hypothesis (*** però vedi theoretical underpinnings...) on the partial order associated with the relation more-general-than. It can be any directed acyclic graph. Also, it is not necessary for the most specific nodes to be singletons (i.e. no single representation trick). Two examples of concept language, which we shall use an illustration throughout the paper, are shown in fig.1 (arrows represent the relation more-general-than).

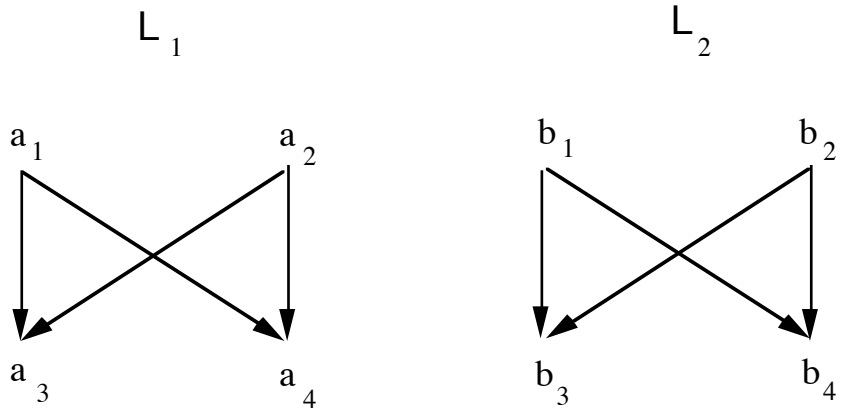


Fig. 1. Two concept languages and their partial orders.

A language $L_1$ is *larger than* a language $L_2$ if the set of concepts expressible in $L_1$ is a proper superset of the set of concepts expressible in $L_2$.

The *product* $L_{1,2}$ of two factor languages $L_1$ and $L_2$ is the set of concepts formed from the conjunctions of concepts from $L_1$ and $L_2$ (examples of product concepts are $a_1b_1$, $a_1b_2$, etc). The number of concepts in the product language is therefore the product of the number of concepts in its factors. Also, a concept $c_{c1',c2'}$ in the language $L_{1,2}$ is more general than (>) another concept $c_{c1'',c2''}$ if and only if $c_1' > c_1''$ and $c_2' > c_2''$.

With n initial languages it is possible to generate $\sum_{k=1,n} n! / (n - k)!\, k! = 2^n - 1$ product languages (see fig. 2). Moreover, if each factor language contains the concept 'any', the relation larger-than over this set of languages can be immediately established, for each product language is larger than any of its factor languages.

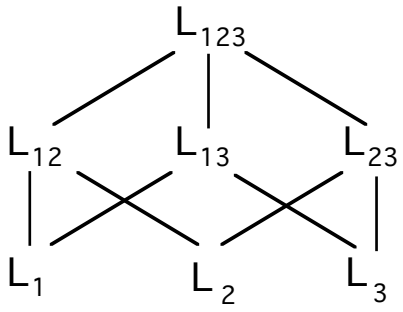


Fig. 2. The graph of product languages with three factor languages.

The learning problem can be stated as follows.

Given

A set of factor concept languages
A set of positive instances.
A set of negative instances.

Incrementally Find

The version spaces in the set of product concept languages that are consistent with data and that contain the smallest number of factors.

## 3. The FCE learning algorithm

In this approach concept learning and language shift are interleaved. We process one instance at a time, using a standard version space approach to induce consistent concepts over *each* language of the current set (initially, the n factor languages) in parallel. As the inductive phase goes on, more and more concept languages are likely to become inconsistent with the data. When every member of the current set of languages has become inconsistent with data, the language shifting algorithm is invoked. It iteratively selects the set of maximally small concept languages that are larger than the current ones (i.e. the two-factored languages, the three-factored languages, etc.) and computes the new version spaces in these languages. It halts when it finds a consistent set of concept languages (i.e. a set in which there is at least one consistent concept language); then it returns control to the inductive algorithm to process additional examples.The whole process is iterated as long as the set of current languages can be further specialised (i.e. until the n-factored language has been generated). We call this algorithm Factored Candidate Elimination (FCE) algorithm. The top-level FCE algorithm is presented in table 1.

Table 1: The top-level FCE algorithm

---

Input: An instance set {I}.
A set of partially ordered concept languages {L} formed by given factor languages and their products.

Output: The version spaces in the set of languages {L} that are consistent with {I} and that contain the smallest number of factors.

Variables: $\{L_S\}$ is a set of unordered languages.
{VS} is a set of version spaces, with $|VS| = |L_S|$.
$\{L_S,VS\}$ is the set of pairs obtained pairing the corresponding elements in $\{L_S\}$ and {VS}.
K is the number of factors contained in each element of $\{L_S\}$.

Procedures: CE(i,l,vs) takes an instance, a concept language and a version space and returns the updated version space.

```
FCE({I},{L})
        Let K=1.
        Let {L_S} be the set of K-factored languages in {L}.
        Let {VS} be a set of empty version spaces.
        For each instance i in {I},
                For each (l_s,vs) in {L_S,VS}
                        Let vs=CE(i,l_s,vs).
                If all the version spaces in {VS} are empty
                        Then Repeat
                                Let K=K+1.
                                Let {L_S} be the set of K-factored languages in {L}.
                                For each l_s in {L_S},
                                        find the new version space vs associated with it.
                                Until at least one vs is not empty.
```

---

The core of the algorithm is the procedure to find the new consistent version spaces in the product languages (in italics in table 1). The difficulty is that the algorithm for inducing concepts over a language (the inductive algorithm) is usually distinct from the algorithm for adding new terms to the language itself (the language-shifting algorithm). In general, the inductive algorithm has to be run again over the instance set after any change made by the language-shifting algorithm ([Utgo86], [Math89], [Paga89], [Wogu89]). In this case, however, in defining the procedure to induce the new consistent concepts after any language shift, we take advantage of the features of the particular inductive learning algorithm considered (i.e. the CE algorithm) and of the properties of language "multiplication". The two key facts are that the CE algorithm makes an explicit use of concept ordering and that concepts in any product language preserve the order of concepts in its factors. This makes it possible to modify the basic CE algorithm with the aim of computing the set of consistent concepts in a product language as a function of some appropriate concept sets induced in its factors rather than by reprocessing the instances already seen.

The concept sets computed in each factor language which will be utilized during language shift are the following. First, for each language we compute the set S*. S*

contains the most specific concepts in the language that cover *all* positive examples, regardless of whether or not they include any negative examples. Second, for each language *and each negative example*, we compute the set G*. G* contains the most general concepts in the language that do not cover the negative example, regardless of whether or not they include all positive examples.

These operations can be better illustrated with an example. Let us suppose that we begin with the two concept languages introduced above - $L_1$ and $L_2$ - enriched with the two dummy super-concepts 'any-a' and 'any-b' ('a' and 'b' for short). Let us suppose the system is given one positive example - the Jack of spades - and two negative examples - the Jack of hearts and the Two of spades. We compute the two corresponding version spaces (one for each language), the sets S* (one for each language), and the sets G* (one for each language and for each negative example) in parallel. In particular, the sets S* and G* can be immediately determined, given the ordering over each language's members. The inductive phase is pictured in fig.3 (f stands for face, b for black,etc).

The three instances cause both of the version spaces to reduce to the empty set. The next step is therefore to shift to the set of maximally small concept languages that are larger than $L_1$ and $L_2$ (in this case the product $L_{12}$) and check to see if it contains any concepts consistent with data. The problem of finding the version space in the language $L_{12}$ can be subdivided into the two tasks of finding the lower boundary set $S_{12}$ (i.e. the set of the most specific concepts in $L_{12}$ that are consistent with data) and the upper boundary set $G_{12}$ (i.e. the set of the most general concepts in $L_{12}$ that are consistent with data).